# CALICO Fall '22

November 26th, 2022
180 Minutes
12+9 Problems
Problem Packet by CALICO

# Table of Contents

# Problem 1: she shells book shelves by the she shelf
3 Points

Problem ID: `bookshelf`
Rank: `1`

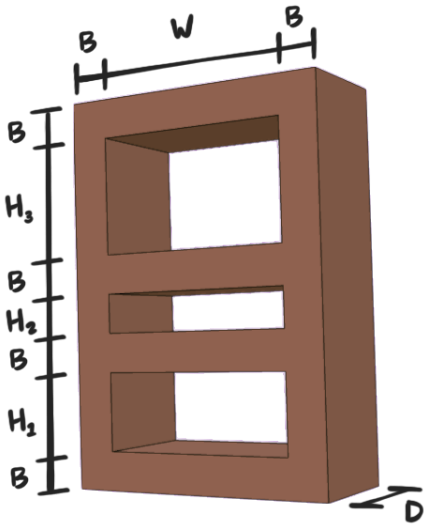
## Introduction

Your graduate "degree" in bookshelves from bookshelf school has finally paid off! After years of toiling over mortise joint design, applying for marquetry internships, and practicing wood impregnation techniques, you're finally working your dream job at JKEA, where you've been spending your days designing premier wooden bookshelves for clients.

There's just one problem: your trusty look-at-it-and-wing-it approach for estimating material costs has led to exorbitant receipts and unhappy customers! After dozens of complaints and multiple lawsuits, your manager has had enough: one more complaint, and you're out of a job. You're already facing a pending fraud investigation into your faked graduate degree, so you need to get your act together—fast!

## Problem Statement

You are given a design for a vertical bookshelf made of wooden boards with a thickness of $\mathbf{B}$ inches. The design has $\mathbf{N}$ vertically-stacked rectangular shelves with boards on each side, with adjacent shelves sharing the same board. The shelves are all $\mathbf{W}$ inches wide (not including the boards) and have heights given by the sequence $\mathbf{H_1}$, $\mathbf{H_2}$, …, $\mathbf{H_N}$, in inches. The bookshelf also has a depth of $\mathbf{D}$ inches but no front or back. Given this design, find the total volume of wood needed to construct the bookshelf, in cubic inches.




*Note: This problem—alongside **all other problems in this contest**—has templates available in Python, Java, and C++! You can find them in the contest.zip provided at the start of the contest. Templates parse the input into a neat function to fill out, so you can jump right into problem solving!*

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains four space-separated integers $\mathbf{N}$ $\mathbf{B}$ $\mathbf{W}$ $\mathbf{D}$ where:
  - $\mathbf{N}$ denotes the number of shelves in the bookshelf.
  - $\mathbf{B}$ denotes the thickness of the boards, in inches.
  - $\mathbf{W}$ denotes the width of the shelves, in inches.
  - $\mathbf{D}$ denotes the depth of the bookshelf, in inches.
- The second line contains a space-separated sequence of $\mathbf{N}$ integers $\mathbf{H_1}$, $\mathbf{H_2}$, …, $\mathbf{H_N}$ denoting the height of each of the bookshelf's shelves, in inches.

# Output Format

For each test case, output a single line containing the total volume of wood needed to construct the bookshelf, in cubic inches.

# Constraints

$1 \le \mathbf{T} \le 100$

$1 \le \mathbf{N}, \mathbf{B}, \mathbf{W}, \mathbf{D} \le 50$

$1 \le \mathbf{H_i} \le 50$

# Sample Test Cases

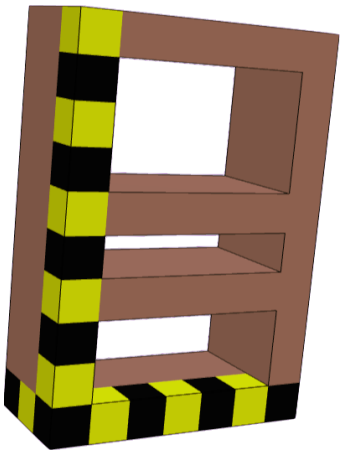
**Sample Input** *Download*

```
3
3 1 5 3
2 1 3
5 2 20 15
3 3 9 1 21
1 3 1 1
5
```

**Sample Output** *Download*

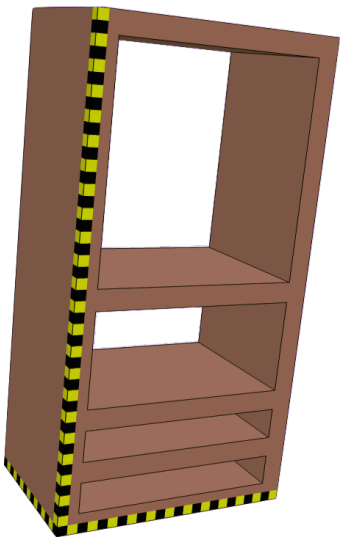
```
120
6540
72
```

**Sample Explanations**



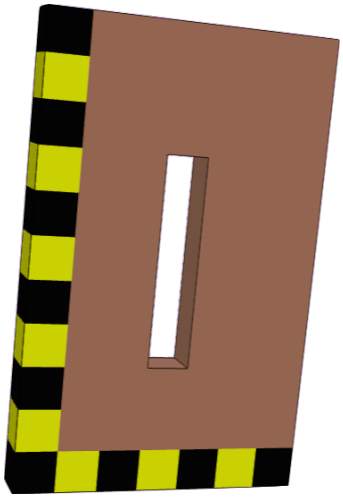
Test Case #1:
The bookshelf described looks like this, and has a total volume of 120 cubic inches. The left and right sides of the bookshelf each contribute 1 × 10 × 3 = 30 cubic inches of volume, and each horizontal board contributes 5 × 1 × 3 = 15 cubic inches of volume. 2 × 30 + 4 × 15 = 120 cubic inches of volume total.

Note that while the width of each shelf $\mathbf{W}$ is 5 inches, the width of the entire bookshelf is actually 7 inches.



Test Case #2:
The bookshelf described looks like this, and has a total volume of 6540 cubic inches.

Although this bookshelf seems much larger than the other two samples provided, it would actually only be just over 4 feet (1.2 meters) tall!



Test Case #3:
The bookshelf described looks like this, and has a total volume of 72 cubic inches. Note how boards of thickness $\mathbf{B} = 3$ surround the only shelf on all sides.

Admittedly, the bookshelf only having a depth of $\mathbf{D} = 1$ makes this more of a door than a proper bookshelf.

# Problem 2: Office of Business Contracts and Brand Protection
# 3 Points

Problem ID: `rso`
Rank: `1`

## Introduction

Registered Student Organizations (RSOs) at the University of California like CALICO are subject to a strict set of *lame* rules set by the Office of Business Contracts and Brand Protection that dictate what we can and cannot call ourselves. In fact, just last week, they threatened to take away our https://calico.berkeley.edu/ domain! >_< Terrifying! This problem is based on a true story of confusing guidelines and a CALICO identity crisis.

## Problem Statement

Given the year an RSO was registered $\mathbf{Y}$ and the name they registered with $\mathbf{N}$, determine if the RSO's name is valid according to the University's branding guidelines.

The branding guidelines are as follows:

- RSO names must be no longer than 50 characters long.
- RSO names must not begin with any trademarked *words* in any capitalization.
  - The following words are trademarked:
    - "California"
    - "Cal"
    - "Berkeley"
  - **Note that this only applies to *full words*.** For example, "Calendar" is allowed but "Cal" is not.
- RSOs registered prior to but not including the year 2010 are exempt from the trademarked word rule.

*Note: This problem—alongside **all other problems in this contest**—has templates available in Python, Java, and C++! You can find them in the contest.zip provided at the start of the contest. Templates parse the input into a neat function to fill out, so you can jump right into problem solving!*

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow.
For each test case:

- The first line contains an integer $\mathbf{Y}$ denoting the year the RSO was established.
- The second line contains a string $\mathbf{N}$ denoting the name the RSO registered with.

# Output Format

For each test case, output a single line containing `VALID` if the RSO's name is valid or `INVALID` otherwise.

# Constraints

$1 \leq \mathbf{T} \leq 100$
$1868 \leq \mathbf{Y} \leq 2022$ (The University was founded in 1868)
$1 \leq |\mathbf{N}| \leq 100$
$\mathbf{N}$ only contains letters from the lowercase or uppercase English alphabet:
`abcdefghijklmnopqrstuvwxyz`
`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
$\mathbf{N}$ can also contain spaces, but it is guaranteed that it will never start or end with a space.

# Sample Test Cases

### Main Sample Input

*Download*

```
8
2021
California Informatics Competition
2021
CALICO Informatics Competition
2022
Competitive Coding at Berkeley
1989
Cal Animage Alpha
2011
Berkeley Math Tournament
2011
Math Tournament at Berkeley
2005
BERKELEY BUSINESS SOCIETY
2003
Saving The World By Overloading It With Fun Brigade
```

### Main Sample Output

*Download*

```
INVALID
VALID
VALID
VALID
INVALID
VALID
VALID
INVALID
```

### Sample Explanations

For test case #1, the name begins with the word `California` which is trademarked.

For test case #3, the name contains the word `Berkeley` which is trademarked, but this is allowed since it does not start with a trademarked word.

For test case #7, the name begins with the word `BERKELEY` which is trademarked. However, clubs before 2010 are exempt. Keep in mind trademarked words can be in any capitalization.

For test case #8, the name is longer than 50 characters.

# Problem 3: This problem was not brought to you by jane street
# 3 Points

Problem ID: `paint`
Rank: `1`

## Introduction

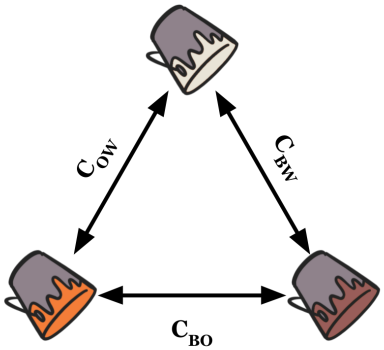
After successfully purchasing your offshore paint assets, you want to consolidate your funds to centralize your investment portfolio into pigment futures. Following the advice of world-renowned financial guru 🅱️onald 🅱️uck, you know that converting all of your paint into a single color is the smartest way to go. However, your math skills aren't up to par and so you resort to a computer program to make your decisions.

## Problem Statement

Given $\mathbf{W}$ buckets of white paint, $\mathbf{O}$ buckets of orange paint, and $\mathbf{B}$ buckets of brown paint, find the minimum cost to convert all of the paint into a single color.

It costs $\mathbf{C_{OW}}$ dollars to convert between a bucket of orange and white paint, $\mathbf{C_{BO}}$ dollars to convert between a bucket of brown and orange paint, and $\mathbf{C_{BW}}$ dollars to convert between a bucket of brown and white paint. All costs are bidirectional: it costs $\mathbf{C_{OW}}$ to convert a bucket of orange paint to white paint, and likewise a bucket of white paint to orange paint. Since the paint is very robust, you can convert the same bucket of paint as many times as you like, including back to previous colors.

As you are on a budget, you must convert all paint to a single color using as little money as possible.




*Note: This problem—alongside **all other problems in this contest**—has templates available in Python, Java, and C++! You can find them in the contest.zip provided at the start of the contest. Templates parse the input into a neat function to fill out, so you can jump right into problem solving!*

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow.
For each test case:

- The first line contains 6 space-separated integers $\mathbf{W}$ $\mathbf{O}$ $\mathbf{B}$ $\mathbf{C_{OW}}$ $\mathbf{C_{BO}}$ $\mathbf{C_{BW}}$ where:
  - $\mathbf{W}$ denotes the number of buckets of white paint you have.
  - $\mathbf{O}$ denotes the number of buckets of orange paint you have.
  - $\mathbf{B}$ denotes the number of buckets of brown paint you have.
  - $\mathbf{C_{OW}}$ denotes the cost to convert between a bucket of orange and white paint.
  - $\mathbf{C_{BO}}$ denotes the cost to convert between a bucket of brown and orange paint.
  - $\mathbf{C_{BW}}$ denotes the cost to convert between a bucket of brown and white paint.

# Output Format

For each test case, output a single line containing the minimum cost to convert all of the paint into a single color.

# Constraints

$1 \le \mathbf{T} \le 100$
$0 \le \mathbf{W}, \mathbf{O}, \mathbf{B} \le 100$
$1 \le \mathbf{C_{OW}}, \mathbf{C_{BO}}, \mathbf{C_{BW}} \le 100$

# Sample Test Cases

**Sample Input** *Download*

```
3
30 0 0 2 5 10
21 9 10 4 1 2
9 6 6 50 4 8
```

**Sample Output** *Download*

```
0
47
96
```

**Sample Explanations**

Test Case #1:
Since all of the paint is white, we do not require any conversions, so our cost is 0 dollars.

Test Case #2:
To achieve our minimum cost, we must convert all of our paint into white paint. We can convert our brown into white paint for $10 \times 2 = 20$ dollars. However, we must convert our orange paint to brown paint first, before then turning it into white paint. This costs $9 \times (1 + 2) = 27$ dollars, for a total cost of 47 dollars.

Test Case #3:
The minimum cost involves converting all of our paint into brown paint. We convert our white paint into brown paint for $9 \times 8 = 72$ dollars and our orange paint directly for $6 \times 4 = 24$ dollars, for a total cost of 96 dollars.

# Problem 4: Water Bottles
# 4+3=7 Points

Problem ID: `bottles`
Rank: `2+3`

## Introduction

Berkeley students living in some parts of the Foothill residential complex source most potable water from a communal water dispenser. This problem is inspired by an everyday technique we use to reduce the awkwardness of delaying people behind us in line! The real dispenser outputs 2 liters per minute, but we'll say 1 per minute for this problem to make things simpler.

## Problem Statement

$\mathbf{N}$ students numbered 1, ..., $\mathbf{N}$ are lined up at a water dispenser that dispenses water at a constant rate of 1 liter per minute. The $i^{\text{th}}$ student has an empty bottle with capacity $\mathbf{C}_i$ liters that they begin to fill immediately after the previous student has finished (formally, the $i^{\text{th}}$ student begins refilling when all $j^{\text{th}}$ students for which $j < i$ finish refilling).

We define the *wait time* of a student as the **total** time they have to wait until their bottle **finishes** refilling. The students ask you to reorder the line into a new permutation ($a_1$, ..., $a_{\mathbf{N}}$) of the students' numbers such that the students' total wait time is minimized.

**For the bonus test set only**, the students also require a tiebreaker: If multiple permutations result in the minimum total wait time, choose any permutation that minimizes the number of students moved to a new position. Formally, minimize the number of indices $i$ for which $a_i \neq i$. If there are multiple permutations that accomplish this, you may choose any.

# Input Format

The first line of the input contains an integer $\mathbf{T}$, denoting the number of test cases that follow.
For each test case:

- The first line contains a positive integer $\mathbf{N}$ denoting the number of students in line.
- The second line contains a sequence of $\mathbf{N}$ positive integers $\mathbf{C}_1$, ..., $\mathbf{C}_\mathbf{N}$, denoting the bottle capacities in liters.

# Output Format

For each test case, output the following two lines:

- On the first line, output the minimum total wait time in minutes.
- On the second line, output $\mathbf{N}$ integers $a_1$, ..., $a_\mathbf{N}$ ($1 \le a_i \le \mathbf{N}$) where $a_i$ is the new index of the $i^{\text{th}}$ student from the front of the original line. If there are multiple permutations that satisfy all criteria, you may output any.

# Constraints

$1 \le \mathbf{T} \le 100$

**Main Test Set**

$1 \le \mathbf{N} \le 10^3$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed $10^3$.
$1 \le \mathbf{C}_i \le 10^3$
All capacities $\mathbf{C}_i$ **are distinct**.
There is guaranteed to exist exactly one optimal permutation for each test case.

**Bonus Test Set**

$1 \le \mathbf{N} \le 10^5$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed $10^5$.
$1 \le \mathbf{C}_i \le 10^9$
The capacities $\mathbf{C}_i$ **are not guaranteed to be distinct**.

# Sample Test Cases

**Main Sample Input** *Download*

```
1
3
5 1 2
```

**Main Sample Output** *Download*

```
12
2 3 1
```

**Main Sample Explanations**

The optimal permutation rearranges the line into the order (2, 3, 1).

1. Student 2 is first in the new line. They have a 1 L bottle and spend 1 minute refilling it.
2. Student 3 is second. They have to wait 1 minute and then spend 2 minutes refilling their own bottle, finishing after 3 minutes.
3. Student 1 is third. They have to wait 3 minutes and then spend 5 minutes refilling their own bottle, finishing after 8 minutes.

The total wait time is (1 + 3 + 8) min = 12 minutes, and it can be shown that no other permutation results in a total less than or equal to than 12 minutes.

**Bonus Sample Input** *Download*

```
2
3
2 1 1
4
2 2 1 1
```

**Bonus Sample Output** *Download*

```
7
3 2 1
13
3 4 1 2
```

*Note that this is one of many possible correct outputs. If there are multiple solutions, you may output any of them.*

**Bonus Sample Explanations**

For test case #1, the permutations (2, 3, 1) and (3, 2, 1) both achieve the minimum total wait time of 7 minutes. However, (2, 3, 1) moves three students while (3, 2, 1) moves only two, so (3, 2, 1) is preferred.

For test case #2, the permutation (3, 4, 1, 2) moves all students and results in a total wait time of 13 minutes, which can be shown to be the minimum. There are also three other permutations that result in a total wait time of 13 minutes and move all students: (3, 4, 2, 1), (4, 3, 1, 2), and (4, 3, 2, 1). The tiebreaker is thus inconclusive—any of these permutations are acceptable.

# Problem 5: Let's Get this Bread
# 4+2+2+1=9 Points

Problem ID: `bread`
Rank: `2+2+3+3`

**Speedrun Bounty: The first 2 (different) teams to solve either the bonus A or bonus B test sets will win a $5 Panera Bread Gift Card (or other bakery gift card of your choice)!**

## Introduction

I love bread. I love the plush feeling of dough as it's kneaded around and around, the buttery aroma that envelops me moments before I take a bite, the sensation of gluten strands shredding across the front of my teeth, the feeling of bliss decomposing within me as enzymes pounce and tear at the mush that remains, the adrenaline rush as I reach for another loaf I did not pay for, the sense of anticipation that heightens as I lean in for my next bite—I love bread.

## Problem Statement

There are $\mathbf{N}$ days remaining in the semester. Each day, there are $\mathbf{B_1}$, $\mathbf{B_2}$, …, $\mathbf{B_N}$ loaves of bread at the cafeteria. Find the maximum loaves of bread you can eat before the semester ends.

You can access bread at the cafeteria by using up to $\mathbf{K}$ meal cards you've "found on campus," each of which can only be swiped once. When you swipe a meal card at the cafeteria, it activates for the next $\mathbf{D}$ days, including the day that you swipe. You can't swipe any other meal cards during this period (sorry we don't make the rules). If you swipe with less than $\mathbf{D}$ days left in semester, your card will be activated until the semester ends.

Each day you have access to the cafeteria, you'll eat all the available bread. However, if you try to eat on days where $\mathbf{B_i} = 0$, you'll become indignant at the lack of bread supply and **forfeit the remaining activation period** out of protest—refusing any bread you could've eaten.

Unfortunately, your tragic combination of long-term planning and short-term memory means that you'll come to the cafeteria whenever you have an active meal card, **even if you know there'll be no bread available**.

*Note: For the main test set and bonus test set A, $\boldsymbol{K} = 1$ (you have only 1 meal card to swipe).*

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains three space-separated integers $\mathbf{N}$ $\mathbf{K}$ $\mathbf{D}$, where:
  - $\mathbf{N}$ denotes the number of days in the semester.
  - $\mathbf{K}$ denotes the number of meal cards you have.
  - $\mathbf{D}$ denotes the number of days a meal card will be activated for after swiping.
- The next line contains $\mathbf{N}$ space-separated integers $\mathbf{B_1}$, $\mathbf{B_2}$, ..., $\mathbf{B_N}$ denoting the number of bread loaves available at the cafeteria on each day.

# Output Format

For each test case, output a single line containing the maximum loaves of bread you can obtain throughout the semester.

# Constraints

*Note: The test sets for this problem **are not necessarily cumulative**—that is, a solution that passes a test set may not necessarily solve all test sets before it! We encourage you to submit your solution to any test set you believe you can pass.*

$1 \le \mathbf{T} \le 100$
$0 \le \mathbf{B_i} \le 10^4$

**Main Test Set**

$\mathbf{K} = 1$
$1 \le \mathbf{N}, \mathbf{D} \le 50$

**Bonus Test Set A**

$\mathbf{K} = 1$
$1 \le \mathbf{N}, \mathbf{D} \le 10^5$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed $10^5$.

**Bonus Test Set B**

$1 \le \mathbf{N}, \mathbf{K}, \mathbf{D} \le 50$

**Bonus Test Set C**

$1 \le \mathbf{N}, \mathbf{K}, \mathbf{D} \le 10^5$
The sum of $\mathbf{NK}$ across all test cases in a test file does not exceed $10^5$.

# Sample Test Cases

### Main + Bonus A Sample Input

*Download*

```
4
4 1 1
1 0 3 2
10 1 3
3 5 0 9 0 2 3 3 2 3
15 1 25
88 82 0 46 15 66 75 49 83 51 21 70 54 69 45
12 1 4
3 1 4 1 5 9 2 6 5 3 5 8
```

### Main + Bonus A Sample Output

*Download*

```
3
9
644
22
```

### Main + Bonus A Sample Explanations

Test Case #1:
Since you have one card that activates for $\mathbf{D}$ = 1 days when swiped, swiping your meal card on the third day will allow you to eat 3 loaves of bread, the highest possible amount.

Test Case #2:
With one available card that activates for $\mathbf{D}$ = 3 days when swiped, swiping on the fourth day will allow you to eat 9 loaves of bread, the highest possible amount. Note that despite refusing the 2 loaves on the sixth day out of protest (since $\mathbf{B_5}$ = 0), this is still the highest amount you can eat.

Test Case #3:
Even though $\mathbf{D}$ = 25, there are only $\mathbf{N}$ = 15 days left in the semester. So, swiping your meal card on the fourth day will allow you to eat 644 loaves of bread during the remaining 12 days of the semester, the highest possible amount. Note that swiping your meal card before the fourth day, such as on the first day, will allow you to only eat at most 170 loaves of bread before you see that $\mathbf{B_3}$ = 0 and refuse bread for the rest of the semester.

**Bonus B + Bonus C Sample Input** *Download*

```
3
7 2 3
0 100 0 100 5 6 7
7 3 3
5 8 6 7 0 9 3
9 2 4
0 7 0 3 3 2 3 0 2
```

**Bonus B + Bonus C Sample Output** *Download*

```
118
33
13
```

**Bonus B + Bonus C Sample Explanations**

Test Case #1:
With $\mathbf{K}$ = 2 cards, you can achieve the optimal amount of 118 loaves of bread two ways:

1. Swiping a card on the second day allows you to eat 100 loaves of bread, before seeing that $\mathbf{B_3}$ = 0 and refusing bread for the remaining two days. Swiping a card on the fifth day allows you to eat 18 more loaves of bread, for 118 loaves total.
2. Swiping a card on the fourth day allows you to eat 111 loaves of bread. Swiping on the seventh day allows you to eat 7 more loaves of bread, for 118 loaves total.

Test Case #2:
Using 2 of your $\mathbf{K}$ = 3 available cards on the second and sixth day allows you to eat 21 and 12 loaves of bread, respectively, for a total of 33 loaves—the highest possible amount.

Test Case #3:
Swiping on the fourth and ninth day allows you to eat 11 and 2 loaves of bread, respectively, for a total of 13 loaves—the highest possible amount.

# Problem 6: Chainsawsage Man
# 5+3+3+4=15 Points

Problem ID: `sausages`
Rank: `2+2+3+4`

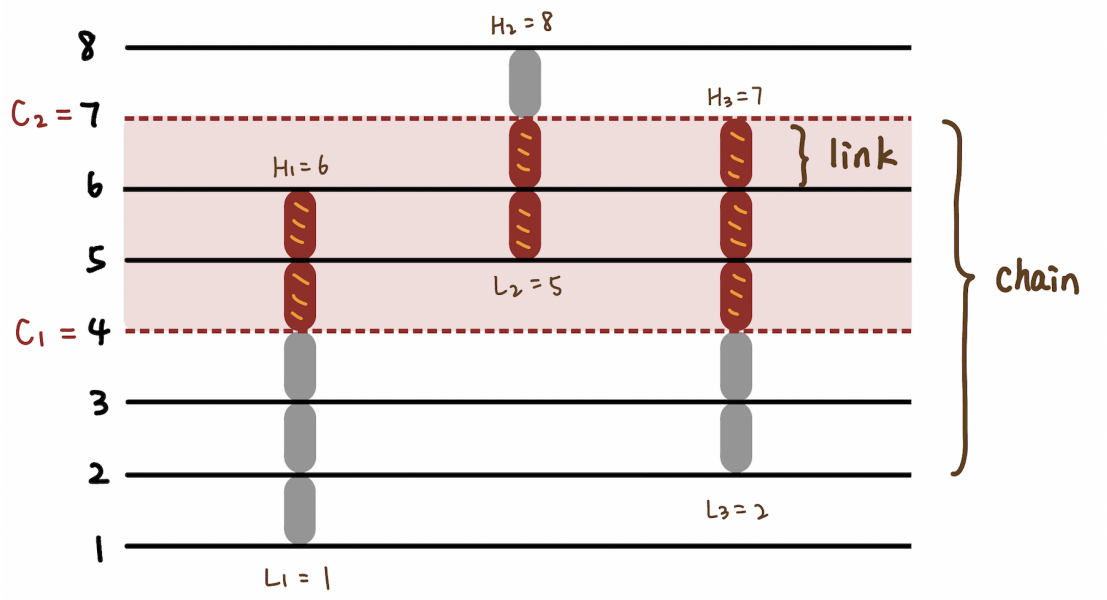
## Introduction

Denji is cutting sausage chains with his chainsaws to cook himself a delicious dinner! These sausages mean a lot to him, as they are what is left of his tale of love, manipulation, and betrayal. He wants to savor the experience by eating an exact amount of sausages, but his chainsaws only cut in a very specific way.

## Problem Statement

You're given $\mathbf{N}$ *chains* of sausages hanging vertically. The heights of the top of each chain are given by integers $\mathbf{H_1}$, $\mathbf{H_2}$, ..., $\mathbf{H_N}$ and the heights of the bottoms of each chain are given by integers $\mathbf{L_1}$, $\mathbf{L_2}$, ..., $\mathbf{L_N}$. Chains consist of individual sausage *links* of length 1 that connect the bottom to the top. Find two integer positions to cut, $c_1$ and $c_2$, so the total number of links between $c_1$ and $c_2$ across all chains is $\mathbf{K}$. If no such cut is possible, output `IMPOSSIBLE`.

For example, cutting at $c_1 = 4$ and $c_2 = 7$ would produce the 7 colored sausage links below:



# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains 2 space-separated integers $\mathbf{N}$ $\mathbf{K}$, where:
  - $\mathbf{N}$ denotes the number of sausage chains.
  - $\mathbf{K}$ denotes the target number of sausage links between your cuts.
- The second line contains $\mathbf{N}$ space-separated integers $\mathbf{H_1}$, $\mathbf{H_2}$, ... , $\mathbf{H_N}$ denoting the heights of the top of each sausage chain.
- The third line contains $\mathbf{N}$ space-separated integers $\mathbf{L_1}$, $\mathbf{L_2}$, ... , $\mathbf{L_N}$ denoting the heights of the bottom of each sausage chain.

# Output Format

For each test case, output a single line containing 2 space-separated integers $c_1$ $c_2$ denoting positions to cut so the total number of sausage links is $\mathbf{K}$. You can output $c_1$ and $c_2$ in any order. If there are multiple valid cuts, output any. If there is no possible cut, output `IMPOSSIBLE`.

# Constraints

$1 \le \mathbf{T} \le 100$
$1 \le \mathbf{K} \le 10^{14}$

**Main Test Set**

$1 \le \mathbf{N} \le 10$
$1 \le \mathbf{L_i} < \mathbf{H_i} \le 10$

**Bonus Test Set 1**

$1 \le \mathbf{N} \le 200$
$1 \le \mathbf{L_i} < \mathbf{H_i} \le 200$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed 1000.
The sum of $max(\mathbf{H_i})$ across all test cases in a test file does not exceed 1000.

**Bonus Test Set 2**

$1 \le \mathbf{N} \le 10^4$
$1 \le \mathbf{L_i} < \mathbf{H_i} \le 10^4$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed $10^5$.
The sum of $max(\mathbf{H_i})$ across all test cases in a test file does not exceed $10^5$.

**Bonus Test Set 3**

**Time Limit: 2 seconds** (this test set only)
$1 \le \mathbf{N} \le 10^5$
$1 \le \mathbf{L_i} < \mathbf{H_i} \le 10^9$
The sum of $\mathbf{N}$ across all test cases in a test file does not exceed $10^5$.
No additional constraints on the sum of $max(\mathbf{H_i})$ across all test cases in a test file.

# Sample Test Cases

**Sample Input** [Download]

```
4
6 8
6 5 8 7 8 3
1 3 5 2 7 1
6 15
6 5 8 7 8 3
1 3 5 2 7 1
7 1
3 3 5 5 5 7 7
2 2 2 4 4 6 6
4 5
6 7 4 7
2 1 2 6
```

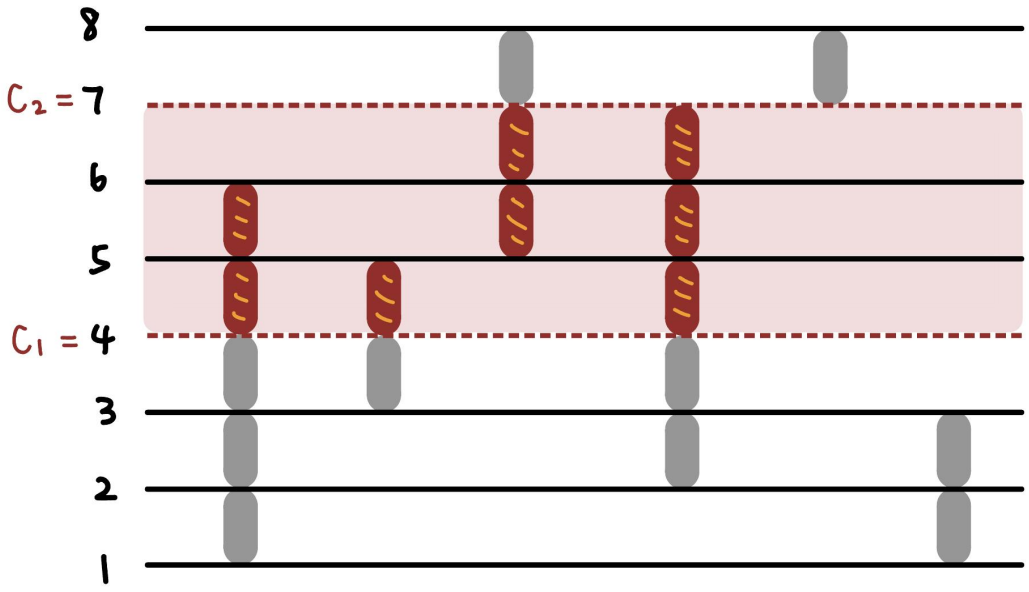
**Sample Output** [Download]

```
4 7
IMPOSSIBLE
3 4
3 5
```

*Note that this is one of many possible correct outputs. If there are multiple solutions, you may output any of them.*

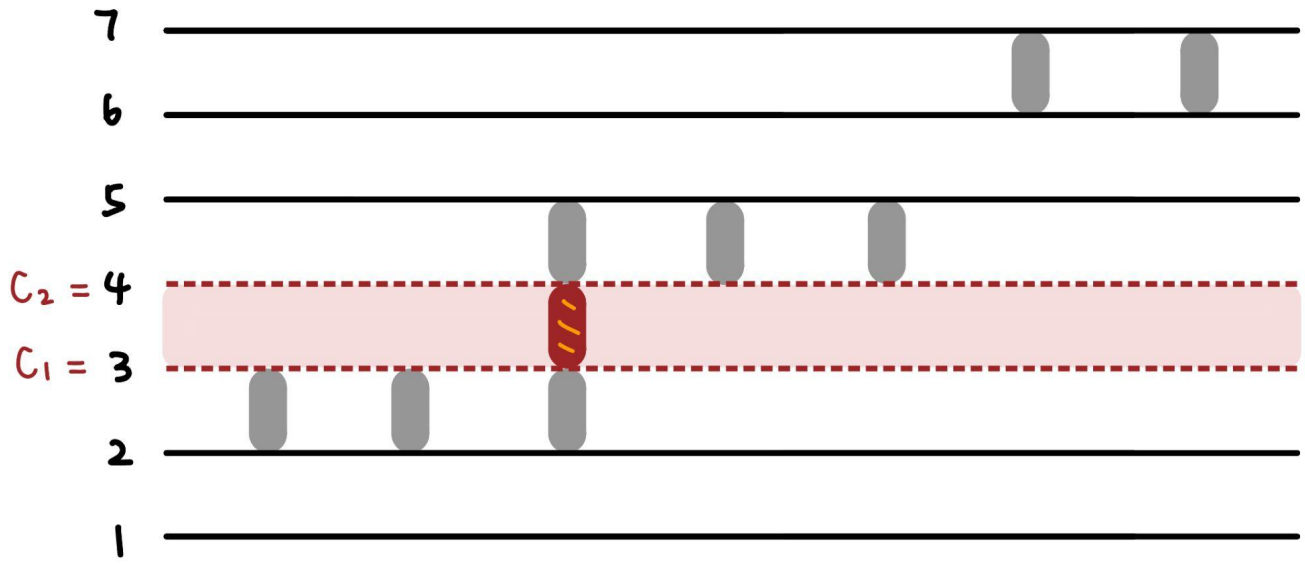
**Sample Explanations**

For test case #1, we have 6 chains positioned as shown in the figure below, and want to find positions to cut that produce exactly 8 links. One way to achieve this is by cutting at heights $c_1$ = 4 and $c_2$ = 7 (or $c_1$ = 7 and $c_2$ = 4). Alternatively, you could also cut at heights 1 and 4.

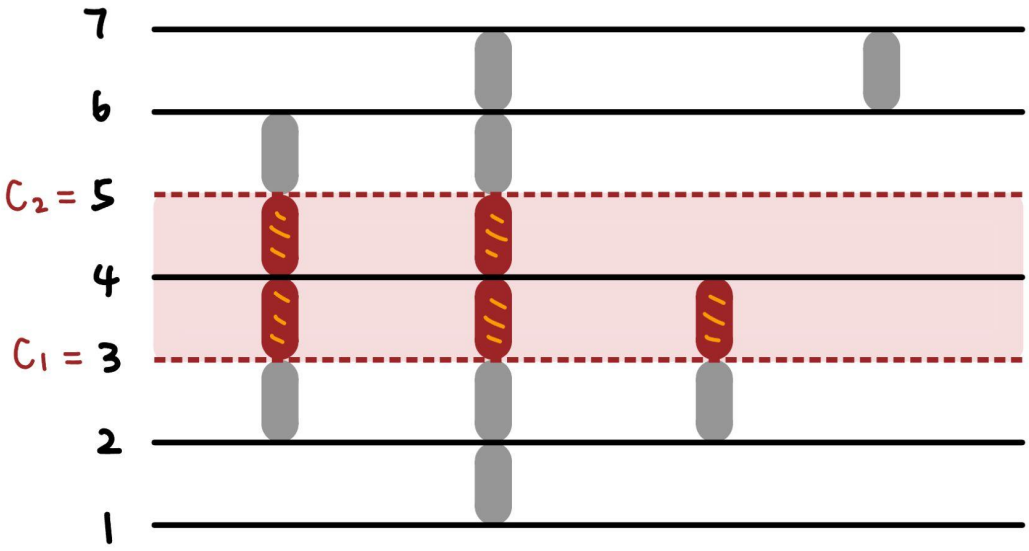



For test case #2, the sausages are positioned the same way as test case #1. However, there's no way to cut the sausages to produce exactly 15 links. (Although 14 and 16 are possible!)

For test case #3, the only way to make cuts that produce exactly 1 link is by cutting the middle link in the third sausage, as shown below.




For test case #4, the only way to make cuts that produce exactly 5 links is by cutting at heights 3 and 5 as shown below. Note that these cut positions are in between sausages and not at endpoints.



# Problem 7: nasin kalama pi toki pona
# 6+2=8 Points

Problem ID: `toki`
Rank: `2+3`

**Code Golf Bounty: The top 2 teams to solve bonus test set 1 with the shortest code size (in bytes) will win a free copy of Toki Pona the Language of Good by Sonja Lang!**

## Introduction

Created in 2001, toki pona is a constructed language whose main philosophy is minimalism: to simplify thoughts and communication. It was designed to be simple to speak, with only 14 phonemes! (one for each letter) Despite this, toki pona has strict rules of how phonemes can be combined to form syllables and words. We can use the linguistic study of syllable structures, phonotactics, to determine what words sound natural in toki pona!

Here is an unofficial dictionary with words you can use to test your program with!

## Problem Statement

Given a word $\mathbf{W}$ made from letters of the English alphabet, determine if it follows the rules of toki pona phonotactics.

In toki pona phonotactics, words are made of one or more valid syllables. All valid syllables follow a *(C)V(*`n`*)* structure: an optional consonant, followed by a mandatory vowel, followed by an optional `n`. Consonants and vowels are single letters from the toki pona alphabet.

The toki pona alphabet only has 14 letters and is a subset of the English alphabet. These are:

- The consonants `m`, `n`, `p`, `t`, `k`, `s`, `w`, `j`, `l` (`n` is a consonant, as well as an optional ending)
- The vowels `a`, `e`, `i`, `o`, `u`

Finally, words must not contain any of these contiguous illegal sequences (including between syllables):

- `wu`, `wo`, `ji`, `ti`, `nn`, or `nm`
- Adjacent vowels such as `aa`, `ei`, `uo`, `aoi`, etc.

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case is described in a single line containing a string $\mathbf{W}$ denoting a word to check.

# Output Format

For each test case, output a single line containing `pona` if $\mathbf{W}$ follows the rules of toki pona phonotactics and `ike` otherwise.

(`pona` and `ike` are the toki pona words for "good" and "bad" respectively)

# Constraints

$1 \le \mathbf{T} \le 100$
$\mathbf{W}$ contains only lowercase letters from the English alphabet:
`abcdefghijklmnopqrstuvwxyz`

### Main Test Set

$1 \le |\mathbf{W}| \le 6$
All valid words in this test set are guaranteed to have at most two syllables.

### Bonus Test Set

$1 \le |\mathbf{W}| \le 10^3$

# Sample Test Cases

**Main Sample Input** *Download*

```
10
a
wan
uta
moku
pona
sinpin
java
jiti
aioli
kek
```

**Main Sample Output** *Download*

```
pona
pona
pona
pona
pona
pona
ike
ike
ike
ike
```

### Main Sample Explanations

For test case #2, `wan` has one valid syllable with a consonant (`w`), vowel (`a`), and `n`. This word means the number one.

For test case #4, `moku` has two valid syllables (`mo` and `ku`). This word means "food" or "to eat".

For test case #7, `java` has `v`, which is not a valid letter in the toki pona alphabet.

For test case #9, `aioli` has adjacent vowels `aio`, which is an illegal sequence. Although not in toki pona, this is what people who try to trick you into eating mayonnaise call mayonnaise.

**Bonus Sample Input** *Download*

```
5
alalalalalalalalalalalalalalan
kepeken
kijetesantakalu
janmisali
wuwojiti
```

**Bonus Sample Output** *Download*

```
pona
pona
pona
ike
ike
```

### Bonus Sample Explanations

For test case #3, `kijetesantakalu` contains seven syllables: `ki`, `je`, `te`, `san`, `ta`, `ka`, and `lu`. This word means any animal from the Procyonidae or Musteloidea family (eg. raccoons).

For test case #5, `wuwojiti` contains four valid syllables that all are also illegal sequences: `wu`, `wo`, `ji`, and `ti`. The meaning of this word is to break toki pona phonotactics.

# Problem 8: your a wizard harry
# 6+6=12 Points

Problem ID: `tower`
Rank: `2+4`

# Introduction

Quickly! You've found yourself whisked into the whimsical world of the hit bestselling fantasy novel, *Herry Bother and the Filibuster's Moan*, and the one and only Herry needs your help! He's been trapped within one of Moldewort's signature creations—an elaborately-constructed circle of towers—and needs to destroy them all in order to escape! Although you're just a humble TikTok ASMR roleplay influencer, you have two things Herry doesn't: a basic understanding of grade-school arithmetic, and the ability to phish Moldevort's tower designs over email. It's up to you to save the day and bring Herry home as fast as possible!

# Problem Statement

You're given $\mathbf{N}$ towers numbered 1, 2, …, $\mathbf{N}$ arranged in a circle. The towers have powers given by the sequence $\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_N}$, where $\mathbf{P_i}$ denotes the power of tower $i$. Adjacent towers are connected by paths of length $\mathbf{D_1}$, $\mathbf{D_2}$, …, $\mathbf{D_N}$, where the distance between tower $i$ and tower $i + 1$ is denoted by $\mathbf{D_i}$, and the distance between tower $\mathbf{N}$ and tower 1 is denoted by $\mathbf{D_N}$. Find the minimum amount of time needed to destroy all the towers.

Herry starts with a power of zero, which increases by one at the end of every hour. Upon arriving at a tower, he can destroy it instantly if he has a power equal to or greater than the power of the tower. Otherwise, he must wait inside the tower until his power matches that of the tower—after which he can destroy the tower instantly.

He can choose to start at any tower he wishes, and can walk at a pace of one unit of distance per hour. Once he begins, he can only travel around the circle in a clockwise direction—despite all of his magic, Herry was never taught as a child how to turn around.

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow.
For each test case:

- The first line contains the single integer $\mathbf{N}$ denoting the number of towers in the circle.
- The second line contains the space-separated sequence of $\mathbf{N}$ integers $\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_N}$ denoting the powers of the towers.
- The third line contains the space-separated sequence of $\mathbf{N}$ integers $\mathbf{D_1}$, $\mathbf{D_2}$, …, $\mathbf{D_N}$ denoting the distances between towers.

# Output Format

For each test case, output a single line containing the minimum number of hours needed to destroy all the towers.

# Constraints

$1 \le \mathbf{T} \le 100$

**Main Test Set**

$1 \le \mathbf{N} \le 100$
$1 \le \mathbf{P_i}, \mathbf{D_i} \le 10^3$
The sum of $\mathbf{N}$ across all test cases does not exceed $10^3$.

**Bonus Test Set 1**

Time Limit: **2 seconds** (This is twice the time of a typical problem because of larger inputs!)
Memory Limit: **500 MB** (This is twice the time of a typical problem because of larger inputs!)
$1 \le \mathbf{N} \le 10^5$
$1 \le \mathbf{P_i}, \mathbf{D_i} \le 10^6$
The sum of $\mathbf{N}$ across all test cases does not exceed $10^5$.
Careful! If you are a Java or C/C++ programmer, be aware that the int variable type may be too small to contain $\mathbf{P_i}$, $\mathbf{D_i}$, or other values needed to solve the problem! Java programmers can use variable types long or float instead, and likewise long long or float for C/C++.

# Sample Test Cases

**Sample Input** *Download*

```
3
3
4 1 9
3 2 3
1
10
17
8
2 9 4 9 5 8 7 1
2 1 3 1 1 2 2 1
```

**Sample Output** *Download*

```
9
10
17
```

### Sample Explanations

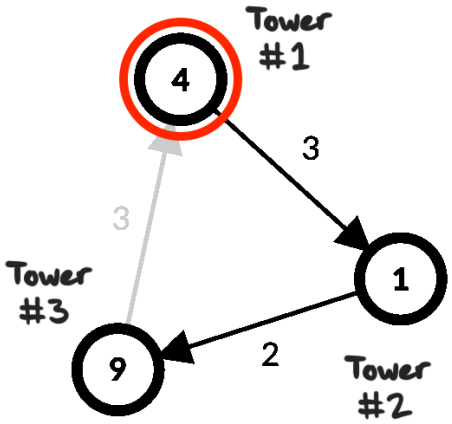
Test Case #1:
The shortest times to destroy all towers from each starting point are:

- Tower #1, with $\mathbf{P_1}$ = 4 (circled in red): 9 hours
- Tower #2, with $\mathbf{P_2}$ = 1: 12 hours
- Tower #3, with $\mathbf{P_3}$ = 9: 15 hours

If Herry starts at tower #1 (circled in red), he can destroy all towers in 9 hours by:

- Destroying the starting tower in 4 hours,
- the next 3 hours after that,
- and the final tower 2 hours after that,

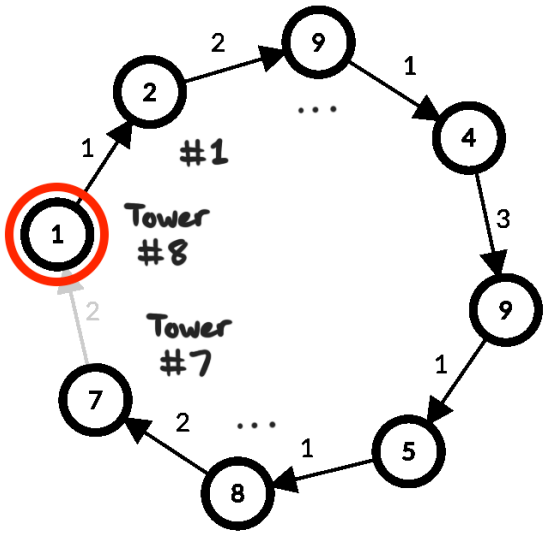
… finishing his journey at tower #3 after 9 hours. Herry ends with a power level of 9—note that Herry does not travel the gray path after destroying the last tower.



Test Case #2:
There is only one starting point—tower #1, with $\mathbf{P}_1 = 10$. After 10 hours, Herry is able to destroy the only remaining tower. Note that Herry does not travel on any paths.

Test Case #3:
The shortest times to destroy all towers from the three best starting points are:

1. Tower #8, with $\mathbf{P}_8 = 1$: 17 hours
2. Tower #3, with $\mathbf{P}_3 = 4$: 18 hours
3. Tower #7, with $\mathbf{P}_7 = 7$: 18 hours




If Herry starts at tower# 8 (circled in solid red), he can destroy all towers in 9 hours by:

- Destroying the starting tower in 1 hour,
- the next 1 hour after that,
- the next 7 hours after that,
- the next 1 hour after that,
- the next 3 hours after that,
- the next 1 hour after that,
- the next 1 hour after that,
- and the final tower 2 hours after that,

… finishing his journey at tower #7 after 17 hours. Herry ends with a power level of 17—note that Herry does not travel the gray path after destroying the last tower.

# Problem 9: SOLIDWORKS is not responding
# 7+2=9 Points

Problem ID: `extrusion`
Rank: `2+3`

## Introduction

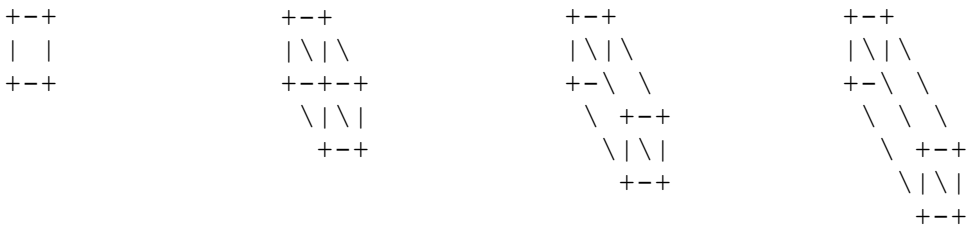
CADence is designing some 3D printed robot parts but the CAD software is not responding! (Probably because it only has 1 second runtime and 256MB RAM) Her parts are all relatively small and simple, and she has designs for them already, but she can't generate previews. Instead of wrestling with SolidWorks, she decides to write her own renderer instead!

## Problem Statement

Given a base with height $\mathbf{H}$ and width $\mathbf{W}$ as a grid of characters with $\mathbf{H}$ rows and $\mathbf{W}$ columns, output an image of the shape that is formed after being extruded to depth $\mathbf{D}$.

The base is a closed 2D shape whose sides all meet at right angles. Every side is either horizontal or vertical. Furthermore, the base has no holes or self-intersections–only a single continuous boundary. In other words, the base is a simple rectilinear polygon.

To extrude, extend the base to construct a 3D prism with **transparent faces but visible edges**. To do this, draw edges from each corner of the base towards the bottom right using backslashes `\` a total of $\mathbf{D}$ times. Each time, draw on top of the existing image. Finally, draw another base at the end to complete the prism. Here are some examples with a simple square base:



Base ($\mathbf{H}$ = 3, $\mathbf{W}$ = 3) $\mathbf{D}$ = 1 $\mathbf{D}$ = 2 $\mathbf{D}$ = 3

Due to the way we draw the extrusion, this results in a "perspective" such that some characters will overlap with each other when drawing. When this happens, draw only the character closer to our perspective in each position.

# Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains three space-separated integers $\mathbf{H}$ $\mathbf{W}$ $\mathbf{D}$ denoting the height of the base, the width of the base, and the depth to be extruded, respectively.
- The next $\mathbf{H}$ lines contain $\mathbf{W}$ characters each. Together, they describe the shape of the base. Each character is one of the following:
  - A space denoting empty space
  - A plus + denoting a corner of the base
  - A pipe | denoting a vertical side of the base
  - A dash – denoting a horizontal side of the base

It is guaranteed that the base is a simple rectilinear polygon as described in the problem statement. Also, no two corners (+) are directly adjacent to one another, every corner connects exactly two sides, and the base extends to all four edges of the grid.

# Output Format

For each test case, output the image created by extruding the base to a depth of $\mathbf{D}$. Use backslashes \ along the diagonals to show edges of the prism created by extrusion.

The judge will accept trailing spaces at the end of lines but not trailing blank lines at the end of each test case. Each line of your output should have at least one non-space character.

Note that your programming language might want you to encode the \ character as \\, as a single \ will be treated as an escape sequence instead.

# Constraints

**Time Limit: 2 seconds** (This is twice the time of a typical problem because of larger inputs!)
$1 \le \mathbf{T} \le 100$
The total size of the correct output in each test file is no more than $10^7$ (10 MB).

**Main Test Set**

$1 \le \mathbf{H}, \mathbf{W}, \mathbf{D} \le 150$

**Bonus Test Set**

$1 \le \mathbf{H}, \mathbf{W}, \mathbf{D} \le 1500$

# Sample Test Cases

**Sample Input** *Download*

```
3
3 3 3
+-+
| |
+-+
7 7 2
  +-+
  | |
+-+ +-+
|     |
+-+ +-+
  | |
  +-+
7 12 1
+----------+
|          |
| +-----+  |
| |     |  |
| +---+|  |
|      |+---+
+------+
```

**Sample Output** *Download*

```
+-+
|\|\
+-\ \
 \ \ \
  \ +-+
   \|\|
    +-+
  +-+
  |\|\
+-+ \-\
|\ \ +-+
+-\ \|\|\
 \|+-+ +-+
  \|\ \ \|
   +-+ +-+
    \|\|
     +-+
+----------+
|\         |\
| +--------+ |
| |\   |\  | |
| |-+----+ | |
| |\| |\-|-+ |
+-|-+---+|  \|
 \|    \|+---+
  +-----+
```

**Sample Explanations**

For test case #1, the base is a square. The bottom right corner of the original base and the edge extruding from the top left corner overlap. Since the edge is closer to our perspective, we output that instead of the corner.

For test case #2, the base has many corners and the image has a lot of overlapping components. Be careful to ensure your perspective is correct.

For test case #3, the base is concave with a big dent and a corner touching a different side.

# Problem 10: Steve was obliterated by a sonically-charged shriek
# 8 Points

Problem ID: `warden`
Rank: `3`

## Introduction

This is the first of a two part problem series! You can find the second part in Problem 12.

You are the Warden—terror of the deep dark and sentinel of the ancient city. A trespassing player named Steve has entered your dominion, and you must destroy them at all costs! Although unable to see the intruder, you can track them by pulsing vibrations through sculk sensors throughout the city. Expose their location and obliterate them with a sonic boom!

## Problem Statement

This is an interactive problem! Communicate with the judge using a series of *pulse* queries and *blast* queries. Using $\mathbf{P}$ = 150 or fewer *pulse*s, find Steve and *blast* them to pass each test case.

The Warden is at (0, 0) on the 2D coordinate plane. Steve is at $(\mathbf{X_S}, \mathbf{Y_S})$, a real number coordinate between **$-10^5$ and $10^5$** predetermined for each test case, but not given to you as input.

To start, you can send a *pulse* to any real number coordinate $(x_p, y_p)$ between **$-10^6$ and $10^6$**. Note that **this area is larger than the area where Steve may be**. When you *pulse*, the judge responds with the Euclidean distance of the following path as a decimal number:

$$\text{Warden} \Rightarrow \text{Pulse Location} \Rightarrow \text{Steve} \Rightarrow \text{Pulse Location} \Rightarrow \text{Warden}$$

In other words, you will receive the value of:

$$d((0, 0), (x_p, y_p)) + d((x_p, y_p), (\mathbf{X_S}, \mathbf{Y_S})) + d((\mathbf{X_S}, \mathbf{Y_S}), (x_p, y_p)) + d((x_p, y_p), (0, 0))$$

where $d$ is the distance function:

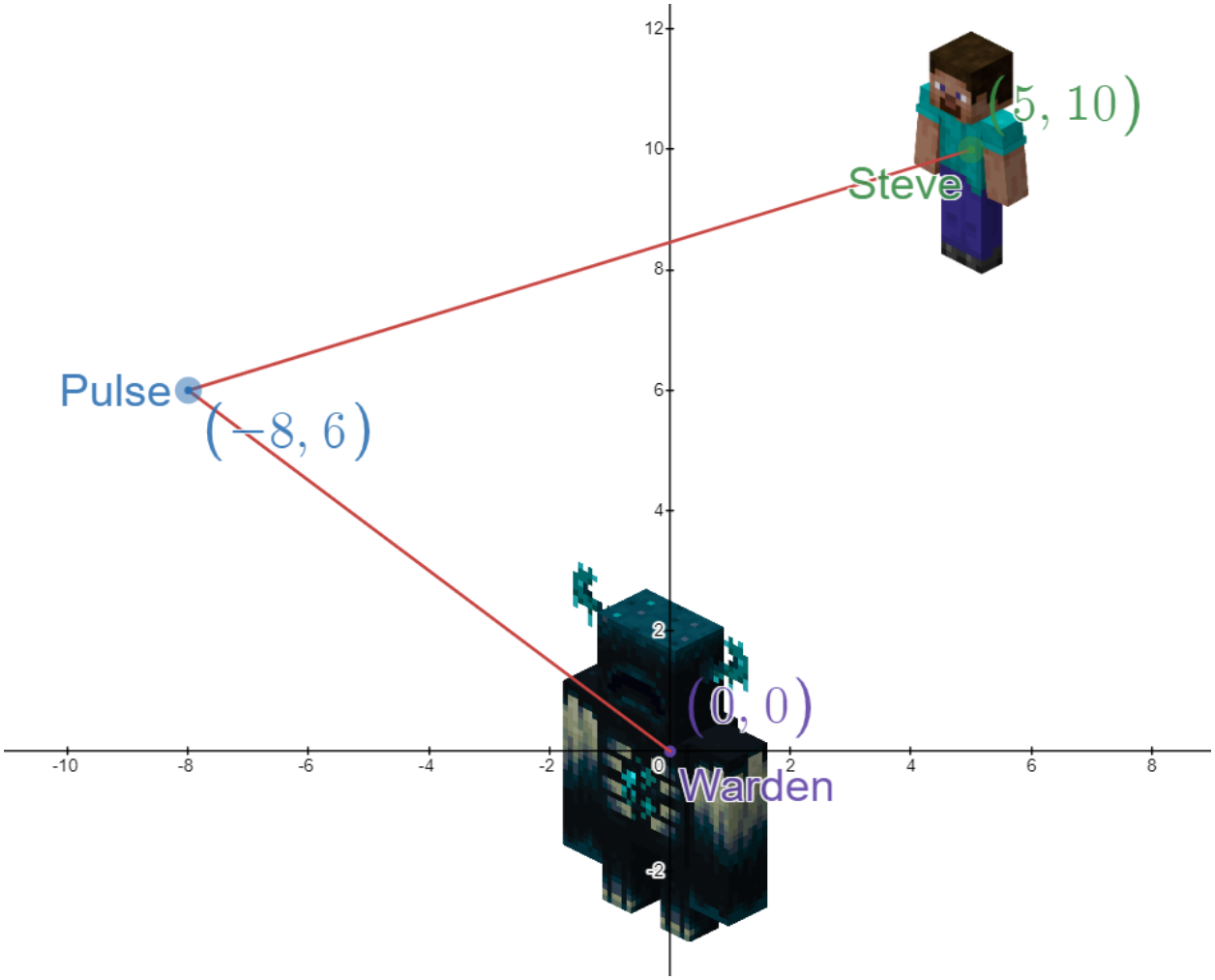
$$d((a, b), (c, d)) = \sqrt{(c - a)^2 + (d - b)^2}$$

After sending up to $\mathbf{P}$ *pulse* queries, you can send a *blast* query to any real number coordinate between **$-10^6$ and $10^6$**. If the distance between your blast location and Steve's location is **at most 100**, you successfully pass the test case. In other words, the condition is:

$$d((x_b, y_b), (\mathbf{X_S}, \mathbf{Y_S})) \leq 100$$

If you are successful, the judge will respond with `CORRECT` and proceed to the next test case. If you are unsuccessful, the judge will respond with `WRONG_ANSWER` and your program should exit to receive a wrong answer verdict.

Here is a Desmos graph for simulating and visualizing pulses and distance calculations! You can use it to try out examples, debug your code, do coordinate math, and more!




*Note: a closed form mathematical solution exists, but we encourage you to seek a solution of a more computational nature. You may find it easier to generalize for the second part that way.*

# Interaction Format

This is an interactive problem! Unlike regular problems, your program and the judge will run simultaneously. Please see the contest guide for more information. Please flush your buffer as instructed by this post when you output, or use our template code that handles it for you. If you run into technical issues with interaction, please let us know with a clarification request!

Begin by reading a single line containing an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

1. Start by making up to $\mathbf{P}$ *pulse* queries. For each query:
   a. First, output a single line containing 3 space separated symbols `P` $x_p$ $y_p$ where:
      - The character `P` signals this is a *pulse* query.
      - The real numbers $x_p$ $y_p$ denote the coordinate to send this pulse to.
   b. Then, read a single line containing a non-negative real number $d$ that denotes the rounded Euclidean distance of this pulse path.
2. Finish by making a single *blast* query as follows:
   a. First, output a single line containing 3 space separated symbols `B` $x_b$ $y_b$ where:
      - The character `B` denotes that this is a *blast* query
      - The real numbers $x_b$ $y_b$ denote the coordinate of the location to blast.
   b. Then, read a single line containing a string that will be `CORRECT` or `WRONG_ANSWER`.
      - If the judge responds with `CORRECT`, you passed this test case.
      - If the judge responds with `WRONG_ANSWER`, your answer is incorrect, and your program should exit to receive a wrong answer verdict.

You can output the real numbers $x_p$, $y_p$, $x_b$, and $y_b$ by expressing them in *decimal notation* like `123.456` or in *scientific notation* like `1.23456e+2` or `1.23456E2`. However, the pulse distance $d$ will always be given in decimal notation, **rounded to $10^{-6}$**.

If your program deviates from the interaction format (e.g. coordinate out of bounds, too many pulse queries, wrong number format, etc.), the judge will send `WRONG_ANSWER`, and your program should exit to receive a wrong answer verdict.

# Constraints

Time Limit: **3 seconds** (I/O can be slow)

$1 \le \mathbf{T} \le 100$

$\mathbf{P} = 150$

$-10^5 \le \mathbf{X_S}, \mathbf{Y_S} \le 10^5$

$-10^6 \le x_p, y_p \le 10^6$

$-10^6 \le x_b, y_b \le 10^6$

# Sample Interaction

*The line spacing here is to emphasize the order in which interaction takes place only. Do not expect or output blank lines between each line of interaction.*

| Sample Input | Sample Output |
|---|---|
| 3 | |
| | P 1 2 |
| 22.360680 | |
| | P -8 6 |
| 47.202941 | |
| | P 13 3.7 |
| 47.398230 | |
| | B 5.7 9.7 |
| CORRECT | |
| | P 12345.6789 98765.4321 |
| 453313.504869 | |
| | B 69420 -42.1 |
| WRONG_ANSWER | |

**Sample Explanations**

The judge begins by outputting `3`, the number of test cases. Before the first interaction, the judge also decides on Steve's location, ($\mathbf{X_S}$ = 5, $\mathbf{Y_S}$ = 10). This is not known by the program.

The program begins the first test case by sending a *pulse* query at ($x_p$ = `1`, $y_p$ = `2`). The judge responds with the rounded distance of the pulse path location: `22.360680`.

Next, the program sends two more pulses at ($x_p$ = `-8`, $y_p$ = `6`) and ($x_p$ = `13`, $y_p$ = `3.7`). The judge responds with `47.202941` and `47.398230`. Note that the program can send pulse locations that are non-integer or negative.

Finally, the program decides it's finished with pulsing even though it only used 3 out of the allowed $\mathbf{P}$ = 150 pulses. It then sends a *blast* query at ($x_b$ = `5.7`, $y_b$ = `9.7`). Although this is not exactly Steve's location of (5, 10), $d((x_b, y_b), (\mathbf{X_S}, \mathbf{Y_S}))$ = $d$((5.7, 9.7), (5, 10)) = 0.761577, which is close enough, so the judge responds with `CORRECT`.

The judge then decides on Steve's location for the next test case, ($\mathbf{X_S}$ = 61926, $\mathbf{Y_S}$ = -18290).

The program makes a *pulse* followed by a *blast* that's too far away from Steve's true location. The judge responds with `WRONG_ANSWER`, and the program exits **before the third test case**.

# Problem 11: 'Tiger's *Tetris* Tournament Trickster
# 10 Points

Problem ID: `tetris`
Rank: `3`

## Introduction

*TETR.IO* is an online multiplayer modern *Tetris* server with millions of players. Qepsi is a Lexington Informatics Tournament (LIT) organizer and *TETR.IO* player currently ranked **top 70 in the world**! She demonstrates her skills in the *Tetris* Game Night, hosted during each iteration of LIT. One time, Qepsi plays so well that CodeTiger, another LIT organizer and *TETR.IO* player, suspects she is cheating by rigging the order of the game's pieces!

## Problem Statement

You're given a sequence of $\mathbf{N}$ pieces $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_\mathbf{N}$ placed from the start of a *Tetris* game. Could these have been placed in a legitimate game, or has the game been tampered with?

*Tetris* has seven pieces: **ZLOSIJT** (in this problem, piece colors are for illustrative purposes only.) The game generates a sequence of contiguous *bags*, each containing exactly one of each piece in a random order. The player receives pieces from these bags one after another, with a new bag beginning after the previous bag finishes. **The generated sequence is not necessarily the same as the given placed sequence!**

Whenever a player receives a piece, they have the option to *place* the piece in the placed sequence or *hold* the piece in their *hold slot*, which is initially empty.

1. If they **place** the piece, it is appended to the end of the placed sequence.
2. If they **hold** the piece, what happens next depends on the hold slot:
    a. If the hold slot is empty, it stores the current piece, and the player receives the next piece without placing anything.
    b. If the hold slot is not empty, the current piece is swapped with the held piece. The player then *places* the piece that was initially held.

We say that the given sequence could have been placed in a legitimate game if there exists both a generated sequence and a series of moves under the above rules that produce it.

## Input Format

The first line of the input contains an integer $\mathbf{T}$ denoting the number of test cases that follow.
For each test case:

- The first line contains an integer $\mathbf{N}$ denoting the number of pieces placed.
- The second line contains a string of uppercase letters $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_\mathbf{N}$, denoting the sequence of **placed** pieces.

## Output Format

For each test case, output a single line containing `YES` if $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_\mathbf{N}$ could have been placed in a legitimate game and `NO` otherwise.

## Constraints

$1 \le \mathbf{T} \le 100$
$1 \le \mathbf{N} \le 10^5$
The sum of $\mathbf{N}$ across all test cases in an input does not exceed $10^5$.
$\mathbf{P}$ contains only letters from the *Tetris* pieces **ZLOSIJT**.

# Sample Test Cases

**Sample Input** *Download*

```
3
3
LIT
14
JOTLIZZJSOTLSO
3
LOL
```

**Sample Output** *Download*

```
YES
YES
NO
```

**Sample Explanations**

Test Case #1:
The pieces could have been placed directly (as in without holds) from the bag **LITZOSJ**. Many other bags would also have allowed this sequence.

Test Case #2:
One possibility is that the first bag was **JOTLIZS**. After placing the first six pieces directly, the **S** would be held, filling a previously empty hold slot. The game could then generate the second bag **ZJSOTLI**, from which the first six pieces are placed again. Instead of placing the **I**, however, the player swaps the held **S** with the **I** and places **S** instead. To place the final **O**, the player could directly place from any third bag starting with **O**.

Test Case #3:
Placing **L** twice would require taking pieces from at least two bags. With only three total pieces, the game could not have generated the second bag yet, indicating tampering.

# Problem 12: Alex was obliterated by a sonically-charged shriek
# 13 Points

Problem ID: `warden2`
Rank: `4`

## Introduction

This is the second of a two part problem series! You can find the first part here. Key differences introduced in this second part are highlighted for emphasis.

You are the Warden—terror of the deep dark and sentinel of the ancient city. After respawning, Steve brings along a friend this time: Alex! You must destroy both of them at all costs! Again, although unable to see the intruders, you can track them by pulsing vibrations through sculk sensors throughout the city. Expose their locations and obliterate them with a sonic boom!

## Problem Statement

This is an interactive problem! Communicate with the judge using a series of *pulse* queries and *blast* queries. Using $\mathbf{P} = 1500$ or fewer *pulse*s, find both Steve and Alex and *blast* them to pass.

The Warden is at (0, 0) on the 2D coordinate plane. Steve is at $(\mathbf{X_S}, \mathbf{Y_S})$ and Alex is at $(\mathbf{X_A}, \mathbf{Y_A})$. Both of these are real number coordinates between $\mathbf{-10^5}$ **and** $\mathbf{10^5}$ predetermined for each test case, but not given to you as input. In an effort to not get blasted together, Steve and Alex are guaranteed to be at least a distance of 1000 apart on each axis.

To start, you can send a *pulse* to any real number coordinate $(x_p, y_p)$ between $\mathbf{-10^6}$ **and** $\mathbf{10^6}$. Note that **this area is larger than the area where Steve and Alex may be**. When you *pulse*, the judge responds with the Euclidean distance of the following path as a decimal number:

$$\text{Warden} \Rightarrow \text{Pulse Location} \Rightarrow \text{Steve} \Rightarrow \text{Pulse Location} \Rightarrow \text{Alex} \Rightarrow \text{Pulse Location} \Rightarrow \text{Warden}$$

In other words, you will receive the value of:

$$d((0, 0), (x_p, y_p)) + d((x_p, y_p), (\mathbf{X_S}, \mathbf{Y_S})) + d((\mathbf{X_S}, \mathbf{Y_S}), (x_p, y_p)) + d((x_p, y_p), (\mathbf{X_A}, \mathbf{Y_A})) + d((\mathbf{X_A}, \mathbf{Y_A}), (x_p, y_p)) + d((x_p, y_p), (0, 0))$$

where $d$ is the distance function:

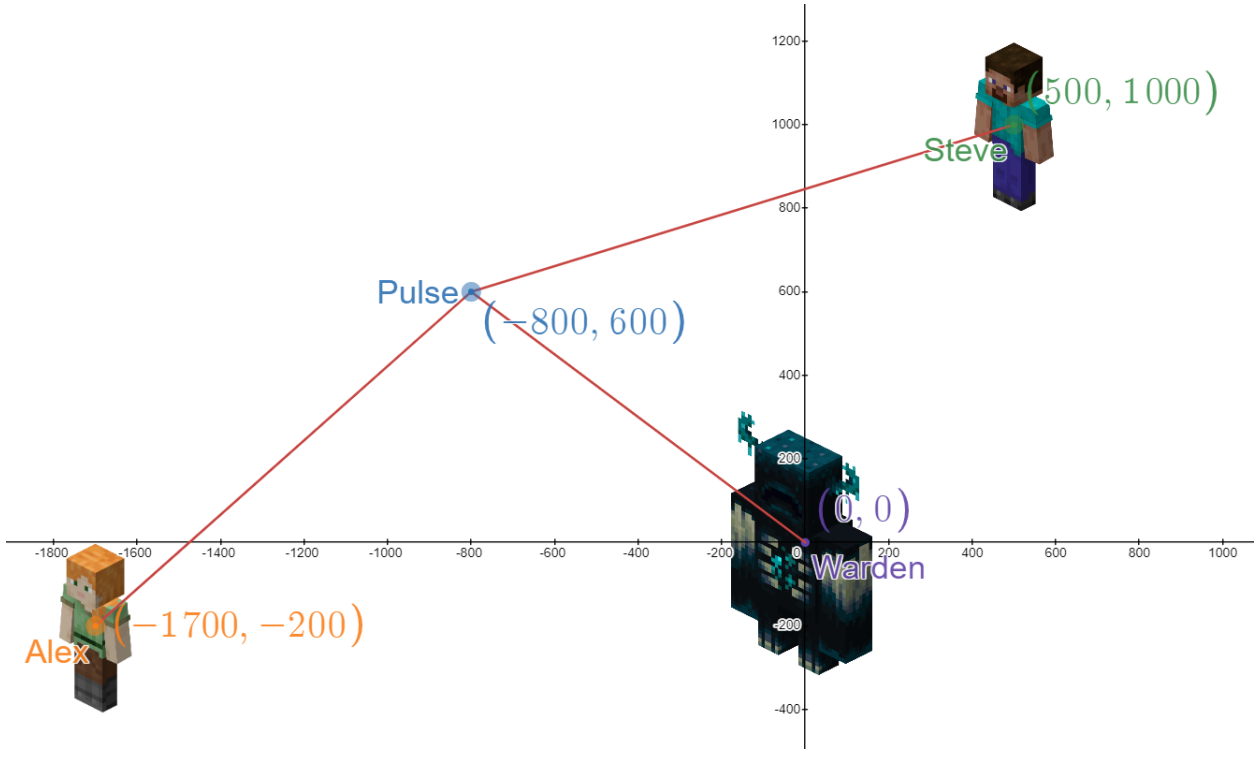
$$d((a, b), (c, d)) = \sqrt{(c - a)^2 + (d - b)^2}$$

After sending up to $\mathbf{P}$ *pulse* queries, you can send a *blast* query to blast at any two real number coordinates between **$-10^6$ and $10^6$**. If the distance between a blast location and Steve's location and the distance between the other blast location and Alex's location are **both at most 100**, you successfully pass the test case. In other words, the condition is:

$$d((x_{b1}, y_{b1}), (\mathbf{X_S}, \mathbf{Y_S})) \le 100 \text{ AND } d((x_{b2}, y_{b2}), (\mathbf{X_A}, \mathbf{Y_A})) \le 100$$
$$\text{OR}$$
$$d((x_{b1}, y_{b1}), (\mathbf{X_A}, \mathbf{Y_A})) \le 100 \text{ AND } d((x_{b2}, y_{b2}), (\mathbf{X_S}, \mathbf{Y_S})) \le 100$$

If you are successful, the judge will respond with `CORRECT` and proceed to the next test case. If you are unsuccessful, the judge will respond with `WRONG_ANSWER` and your program should exit to receive a wrong answer verdict.

Here is a **new** Desmos graph for simulating and visualizing pulses and distance calculations!
Again, you can use it to try out examples, debug your code, do coordinate math, and more!



# Interaction Format

This is an interactive problem! Unlike regular problems, your program and the judge will run simultaneously. Please see the contest guide for more information. Please flush your buffer as instructed by this post when you output, or use our template code that handles it for you. If you run into technical issues with interaction, please let us know with a clarification request!

Begin by reading a single line containing an integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

1. Start by making up to $\mathbf{P}$ *pulse* queries. For each query:
   a. First, output a single line containing 3 space separated symbols `P` $x_p$ $y_p$ where:
      - The character `P` signals this is a *pulse* query.
      - The real numbers $x_p$ $y_p$ denote the coordinate of sculk to send this pulse to.
   b. Then, read a single line containing a non-negative real number $d$ that denotes the rounded Euclidean distance of this pulse path.
2. Finish by making a single *blast* query as follows:
   a. First, output a single line containing 5 space separated symbols `B` $x_{b1}$ $y_{b1}$ $x_{b2}$ $y_{b2}$ where:
      - The character `B` denotes that this is a *blast* query
      - The real numbers $x_{b1}$ $y_{b1}$ $x_{b2}$ $y_{b2}$ denote the coordinates of the locations to blast.
   b. Then, read a single line containing a string that will be `CORRECT` or `WRONG_ANSWER`.
      - If the judge responds with `CORRECT`, you passed this test case.
      - If the judge responds with `WRONG_ANSWER`, your answer is incorrect, and your program should exit to receive a wrong answer verdict.

You can output the real numbers $x_p$, $y_p$, $x_{b1}$ $y_{b1}$ $x_{b2}$, and $y_{b2}$ by expressing them in *decimal notation* like `123.456` or in *scientific notation* like `1.23456e+2` or `1.23456E2`. However, the pulse distance $d$ will always be given in decimal notation, **rounded to $10^{-6}$**.

If your program deviates from the interaction format (e.g. coordinate out of bounds, too many pulse queries, wrong number format, etc.), the judge will send `WRONG_ANSWER`, and your program should exit to receive a wrong answer verdict.

# Constraints

Time Limit: **3 seconds** (I/O can be slow)

$1 \le \mathbf{T} \le 100$

$\mathbf{P} = 1500$

$-10^5 \le \mathbf{X_S}, \mathbf{Y_S}, \mathbf{X_A}, \mathbf{Y_A} \le 10^5$

$-10^6 \le x_p, y_p \le 10^6$

$-10^6 \le x_{b1}, y_{b1}, x_{b2}, y_{b2} \le 10^6$

# Sample Interaction

*The line spacing here is to emphasize the order in which interaction takes place only. Do not expect or output blank lines between each line of interaction.*

| Sample Input | Sample Output |
|---|---|
| 3 | |
| | P 100 200 |
| 5923.885760 | |
| | P -800 600 |
| 7128.613018 | |
| | P 1300 300.7 |
| 10876.748328 | |
| | B 500.7 999.7 -1700.1 -200.1 |
| CORRECT | |
| | P 12345.6789 98765.4321 |
| 846379.903725 | |
| | B 69420 -42.1 1010 0.11 |
| WRONG_ANSWER | |

**Sample Explanations**

The judge begins by outputting `3`, the number of test cases. The judge also decides on Steve's and Alex's locations, ($\mathbf{X_S}$ = 500, $\mathbf{Y_S}$ = 1000) and ($\mathbf{X_A}$, $\mathbf{Y_A}$) = (-1700, -200).

The program begins by sending a *pulse* query at ($x_p$ = `100`, $y_p$ = `200`), to which the judge responds with the rounded distance of the pulse path to that sculk location: `5923.885760`.

Next, the program sends two more pulses at ($x_p$ = `-800`, $y_p$ = `600`) and ($x_p$ = `1300`, $y_p$ = `300.7`). The judge responds with `7128.613018` and `10876.748328`.

Finally, the program sends a *blast* query at ($x_{b1}$ = `500.7`, $y_{b1}$ = `900.7`) and ($x_{b2}$ = `-1700.1`, $y_{b2}$ = `-200.1`). These are not Steve's and Alex's exact locations, but $d$(($x_{b1}$, $y_{b1}$), ($\mathbf{X_S}$, $\mathbf{Y_S}$)) = $d$((500.7, 999.7), (500, 1000)) = 0.761577 and $d$(($x_{b2}$, $y_{b2}$), ($\mathbf{X_A}$, $\mathbf{Y_A}$)) = $d$((-1700.1, -200.1), (-1700, -200)) = 0.141421, which is close enough, so the judge responds with `CORRECT`.

The judge then decides on Steve's and Alex's locations for the next test case, ($\mathbf{X_S}$ = 61926, $\mathbf{Y_S}$ = -18290) and ($\mathbf{X_A}$ = -81928, $\mathbf{Y_A}$ = -73681).

The program makes a *pulse* query followed by a *blast* query that's too far away from Steve and Alex's locations. The judge responds with `WRONG_ANSWER`, and the program exits.